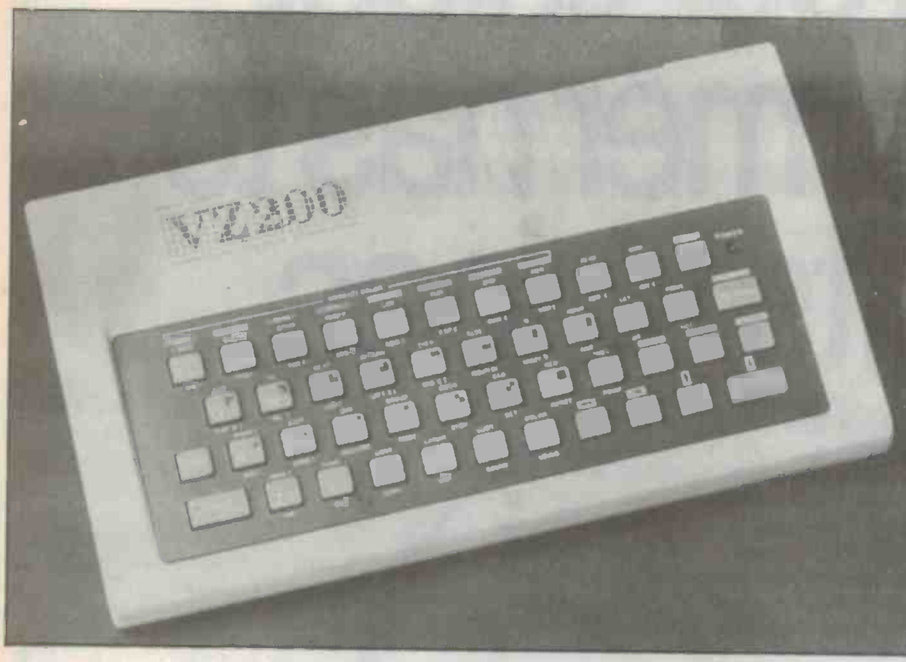

A CHIP-8 INTERPRETER — for VZ200/300

Chris Griffin

How's it going? Did you get the editor from the last article in August '86, typed in, up, and running? If you had any trouble refer to the note at the end of the article. In this article I use the editor to set up the Chip-8 interpreter, to write and run Chip-8 programs. I will also mention details of this particular dialect and show a few simple programs to get you started.

THE CHIP-8 interpreter (Listing 1) is a machine language program which executes instructions beginning at location 8200 (this is in hex — remember!). The interpreter has an 'address space of 4K, meaning that it can only access 4096 bytes of memory. Therefore only three hex digits are required to specify an address. 8200 is referred to as 200 by the Chip-8 interpreter, 54A refers to 854A, etc. So, if from time to time, I drop the leading 8, don't be too bothered about it!

Each Chip-8 instruction consists of two bytes of hexadecimal data — a total of four digits. Between 200 and AFC, the locations in which a program may be stored, there is thus room for about 1150 instructions. You can also use locations (8)000 to (8)1FF to store parts of the program, but never forget that execution is from location 200, so you'll have to use this section of memory for subroutines or shape data.

Chip-8 is a 'what you write is what you get' sort of language in that there is no way to break out of a program that is running, unless you have allowed for this possibility. This is one aspect that could take a little getting used to, but don't worry, you will! The Chip-8 interpreter has in this regard a trade off. A little speed is gained in the sacrifice; and for me, the speed is worth it!

The language of Chip-8 supports only 16 variables, an index register, and a stack pointer (which is rarely used in programs — it is more useful to the interpreter itself!).

The variables, labelled by a 'V', followed by a number (0,1,2...D,E or F), are each one byte long. They can only be used to store numbers in the range 0 to 255, so all operations involving variables are limited in this way. If any extra space is required to store the answer to a calculation, VF is used for the extra piece. (It is called the carry, and is only relevant to a few arithmetic commands. Larger number manipulation is available to a limited degree, using the index register called 'I'. This is a 12-bit number (3 hex digits) and is used to point to memory locations in much the same way that the editor program has a memory pointer. When you store 6B0 in the index register, it points to location 86B0, as might be expected! The index register is an important part of the system as it is used extensively in graphics manipulation; it also allows more than 16 variables to be used by a single program, if desired.

OK, now let's get things up and running!

Getting started

Load your copy of the editor program (ETI August 86 issue), and run it. Then, type in Listing 1 beginning at location 7AE9 (type M7AE9 (cr) P then the data shown in the listing). Check the things typed, to make sure they are correct and type in the following:

(i) M9BDF (cr) P0082 (cr)

This sets the memory pointer to 8200 whenever the editor is run.

(ii) M8EC7 (cr) PE97A (cr)

This connects the Chip-8 interpreter to the editor, allowing it to be activated by pressing XC. 8EC7 is the location which contains the start address for the routine which we want activated by XC — and we store 7AE9, the interpreter start address, here. By the way, locations 8EBF to

8ECD contain the start addresses for all of the X commands (XC through XF), so it's easy to add your own!

(iii) M8200 (cr) PF000 (cr)

A very short Chip-8 program, just to test things out.

Now, save everything. Use OVZCHIP8 (cr) 7AE9 (cr) 8F30 (cr) if you have a tape system, or use BBSAVE "VZCHIPS", 7AE9, 8F30 (cr) if disks are your forte (after saving to disk, you can restart the editor with ?USR(O)).

Let's run the Chip-8 program entered in (iii) above, by pressing XC. The screen should have flashed, and the editor restarted. If it has, so far so good. If not, check that the interpreter you typed in is the same as mine! Tape users will probably have to start all over again!! (This is because B: programs run automatically from tape, but not from disk.) When everything works thus far, read on...

Chip-8 graphics

Graphics takes place on the VZ's mode 1 screen. The individual points are labelled with two coordinates in exactly the same manner as BASIC (except, everything is in hex). Chip-8 allows you to display points (like BASIC), entire shapes (of up to 8 x 16 dots) and line drawings in 256 sizes (although there are some restrictions!) in any combination of colours you care to imagine. (Of course, only four colours can be used at once in this mode — there is little that can be done about this.) An object can be positioned anywhere on the screen, even overlapping another object. Overlapping objects are stored on the screen in exclusive-or form. Table 1 shows the consequences of this in colour mode 0 (COLOR, 0), which is read as: 'if a red object is placed on a blue area of the screen, the overlap is displayed in yellow' etc. Funny idea? Not really! These conditions allow you to remove objects by simply re-displaying them. If we number the colours 0 for green, 1 for yellow, 2 for blue, 3 for red, and change to COLOR, 1 mode the same sort of ideas apply to buff, cyan, magenta and orange.

A collision occurs if the following pairs of colours overlap: 1&1, 2&2, 3&3, 3&1, 3&2. Collisions are registered through an object called 'HIT'. HIT equals 1 means that there has been a collision, HIT equals 0, otherwise. After a graphics command has been executed, HIT is stored in VF (variable F), to allow you to check for collision with Chip-8 instructions.

Shape drawing

A 'SHAPE' is eight dots wide, and between 1 and 16 dots long, and is considered as residing in a grid (see Figure 1 for an example 8 x 9 shape in its grid). Each row of the shape is represented by two bytes of data, that is, four dots to each byte. The colour of each dot can be independently defined using the *number* of the colour that is required.

TABLE 1. COLOUR OVERLAP

Overlapping colours	Green	Yellow	Blue	Red
Green	Green	Yellow	Blue	Red
Yellow	Yellow	Green	Red	Blue
Blue	Blue	Red	Green	Yellow
Red	Red	Blue	Yellow	Green

For the first row of the shape down, we have two green dots (which are in essence *invisible*) five blue dots, and one green dot. The colour codes are 0,0,2,2,2,2,2,0. Group this information into clusters of two digits: 00 22 22 20, then for each cluster, multiply the first digit by 4 and add the second to it, giving 0 A A 8 in our example. The two bytes used to describe this row are thus 0A and A8. Every other row is complete in exactly the same manner and the data stored in a segment of memory.

		B	B	B	B	B	
		B	Y	B	Y	B	
		B	B	B	B	B	
				R			
			R	R	R		
	R	R		R		R	R
				R			
			R		R		
		R				R	

Figure 1. Example of a nine row shape (a robot figure). Each square is filled with the colour that is desired. Those with no colour are green by default, as this behaves invisibly.
Y — yellow colour value is 1
B — blue colour value is 2
R — red colour value is 3
The last row, for example, is 00300030, which is 0C0C in hex.

To put this shape up onto the screen, we set the index register I to point to the first byte of the shape data, and use a SHOW command. From the table of Chip-8 commands (Table 2), it is obvious that the SHOW command is Dxyn, but what does that mean? An example should make this clearer: D456 will show a shape, six rows long, with the top left hand corner at (V4,V5). If we want to display the example shape at (V3,V4), then use the command D349 — the 9 means that our shape is nine rows long.

Let's write up a real Chip-8 program now.

Writing Chip-8 programs

To write a Chip-8 program, simply put the instructions, one after another, in memory from location 200 onwards. Consider the short program that we typed in earlier; pressing XC did nothing much, so what was the Chip-8 program? Well, it consisted of the single instruction F000, which from Table 2, 'jumps back to the editor, or restarts the program if the editor is not found' — in other words: END! So, that's why nothing much happened! For a real program, see Listing 2a. Type this one in (from 8200), and run it XC. You should get the picture we designed earlier in the top left hand corner of the screen. Press a key, and the program ends. Do you understand what went on? The comments given may be of some help! Notice that we didn't need to switch on mode 1 graphics — it's automatic! (Chip-8 operates entirely in this mode.) For more examples, we need more concepts so read on.

Colour registers

The colour register is another VZ/Chip-8 object — like HIT. This, however, is used to store colour data for some commands (Fx29, 8xyD and 8xyE). The register takes on the following values for colours: 00 — invisible or colour 0, 55 — colour 1, AA — colour 2, FF — colour 3. All other values give combinations of these, and are best experimented with! To load the colour register with 55, we could use the following sequence of code. 6F55 FFCC, which says, load VF with 55, then load the colour register with VF. Once the colour is set, we can use 8xyD to plot a point, or Fx29 to draw a number, in the colour that we have defined. Type in and run Listing 2b for an idea of colour register graphics operation.

Joysticks and keyboard

The command ExB4, reads both joysticks at once, and assigns Vx to one of the following values, depending on the joystick position: 00 — nothing, 2E — up, 20 — down, 4D — left, 2C — right, OD — fire. These codes were chosen as they correspond to the cursor control keys on the VZ keyboard. Using ExB3 instead of ExB4 reads the keyboard and allows the result of this command to be treated in an identical manner to the ExB4 command it replaces. The break key returns a value of 01 if it is pressed, so it too can be easily tested for. ▶

Printing out numbers

See Listing 2c for an example of number printing. The Chip-8 interpreter has shape data for the numbers 0,1,2,3...D,E,F automatically built in. All that is required is to retrieve them. The statement Fx29 does just that: retrieves the shape data for the last digit of Vx. If V8 is 7A, F829 retrieves data for the number A, and sets the index register to point to the place where the retrieve data is stored, so that the next display command will show the correct thing. (The data is stored in system memory and will never get in the way of one of your Chip-8 programs.) That's OK for single digit numbers. But what about bigger ones, like 8A, EB etc, or even decimal numbers (for game scores, for instance)?

The process of printing decimal numbers is easy, but fairly long, if you write in Chip-8. See Listing 2d, which repeatedly counts from 0 to 99, for an example. Some important commands are the following.

(i) Fx33, converts Vx to a three digit decimal number, and stores each digit in a different memory location, pointed to by the index register. The hundreds get stored at I, tens at I plus 1, and units at I plus 2, so that if we could load these values into variables, each digit could be displayed in the usual way.

(ii) F265 loads the memory from I, into variables V0, V1 and V2. V0 contains the hundreds, V1 the tens, V2 the units. We can now easily display each digit.

Notice also that the printing process is put in a subroutine at location 228, this saves me repeating the whole process in order to remove the numbers. (Recall: to remove things in Chip-8, simply re-display them.)

TABLE 2 — VZ/CHIP-8 COMMAND SUMMARY

0000 No operation. Does nothing.
00A0 Store I on the subroutine stack.
00A8 Take I off the subroutine stack.
00AE Load I with the subroutine stack pointer.
00C0 Set colour to set 0 (green background).
00C1 Set colour to set 1 (buff background).
00E0 Clear the screen.
00EE Return from a subroutine.
0nnn For nnn larger than 0FF, calls a machine code routine at location 8nnn. Allows user machine code subroutines.
1nnn Go to 8nnn.
2nnn Go sub 8nnn.
3xyy Skip the next instruction if Vx equals yy.
4xyy Skip the next instruction if Vx does not equal yy.
5xy0 Skip the next instruction if Vx equals Vy.
6xyy Load Vx with yy.
7xyy Add yy to Vx.
8xy0 Load Vx with Vy.
8xy1 Load Vx with Vx OR Vy.
8xy2 Load Vx with Vx AND Vy.
8xy3 Load Vx with Vx XOR Vy (exclusive or).
8xy4 Load Vx with Vx plus Vy (the carry is stored in VF).
8xy5 Load Vx with Vx minus Vy (the carry is stored in VF).
8xy6 Load Vx with Vx multiplied by Vy (carry is in VF).
8xyD Plot a point at coordinates (Vx,Vy) with colour as in the colour register.
8xyE Draw a shape with data pointed to by I, of size VF, beginning at the point (Vx,Vy).
9xy0 Skip next instruction if Vx does not equal Vy.
Annn Load I with 8nnn.
Bnnn Go to 8nnn plus V0.
Cxyy Load Vx with a random number ANDed with yy.
Dxyn Show a pattern with data pointed to by I, consisting of n rows with the top left hand corner at (Vx,Vy).
Ex9E Skip the next instruction if Vx equals the key that is down.
ExA1 Skip the next instruction if Vx does not equal the key that is down.
ExB3 Load Vx with the key that is currently down.
ExB4 Load Vx with the present joystick position.
F000 Jump back to the editor or restart the program if no editor is present.
Fx02 Set the sound pitch to Vx.
Px0A Wait for a key to be pressed and load Vx with that key.
Fx18 Beep for Vx cycles.
Fx19 Produce white noise (hiss) for Vx cycles.
Fx1E Add Vx to I.
Fx29 Produce a digit pattern for the last digit of Vx and point I at this pattern (colour is given by colour register).
Fx33 Convert Vx to a decimal number and store each digit in a different byte (100s, 10s, 1s in 3 bytes from 1).
Fx55 Store V0 through Vx to memory pointed to by I (on completion, I is I plus x plus 1).
Fx65 Load V0 through Vx from memory pointed to by I (on completion, I is I plus x plus 1). Opposite of Fx55.
FxCC Load the colour register with Vx.
Any other commands should be avoided — their functions are not defined, but in general, they do *not* represent no operation.

TABLE 3. PITCH/DURATION VALUES FOR SOUND COMMANDS

Pitch	Duration 2	Duration 1	Duration ½	Duration ¼
C 79	79	3C	1E	0F
Db 72	80	40	20	10
D 6C	88	44	22	11
Eb 66	90	48	24	12
E 60	98	4C	26	13
F 5B	A0	50	28	14
Gb 55	AB	55	2B	15
G 50	B5	5B	2D	17
Ab 4C	C0	60	30	18
A 48	CB	66	33	19
Bb 44	D7	6C	36	1B
B 40	E4	72	39	1C
C 3B	F2	79	3B	1E

(Other octaves can be approximated by halving and doubling the pitch and duration values.)

PLOT	LEFT	RIGHT	DOWN	UP	FOUR	TWO	ONE

Figure 2. 8xyE allocation of bits. A '1' in the bit position activates the associated words, eg, PLOT UP and LEFT 5 is 11001101.

How to draw large shapes

8xyE is a command designed to draw large shapes on the graphics screen. Often, the object to be drawn is simple in structure, yet too big for a single 8 x 16 dot shape so under these circumstances, this command is used. 8xyE uses data pointed to by the index register, and also a 'SIZE' value stored in VF, to draw the shape from the point (Vx, Vy). VF equals 1 allows the shape to be drawn exactly as defined. VF equals 2 draws the shape twice the size in both x and y directions, etc. Shape data is given by a series of bytes, from two to as many as required. (Shape data for this command has no maximum length.) The last byte is always 00, required to tell the interpreter when the end has been reached! Each byte, which is made up of eight bits, contains eight pieces of information; Figure 2 gives the key to this. The process of drawing a shape involves directing an invisible cursor about the screen (in eight possible directions), leaving trails as we go if required! A typical instruction to the cursor might be: PLOT UP 2 DOTS, which is coded as 1 0 0 0 1 0 1 0 using 1s and 0s. To get this in hexadecimal form, group data into groups of four : 1000 1010. For each group, convert the binary number into hexadecimal, in this example: 8A.

Example: A square. To draw a square, imagine the following cursor instructions: ▶

LISTING 1.

```
7AE9 = F3 31 FF 8F 3E 09 32 3B
7AF1 = 7B CD 9C 7B 00 00 00 21
7AF9 = FF 7F 22 1C 7F 21 00 82
7B01 = 22 1E 7F 2A 1E 7F 46 23
7B09 = 4E 23 22 1E 7F 78 E6 0F
7B11 = 5F 16 7F C6 80 08 78 1F
7B19 = 1F 1F E6 1E C6 2E 6F 26
7B21 = 7B 08 47 7E 23 6E 67 CD
7B29 = 7D 7B 18 D7 E9 7B 4E 7B
7B31 = 61 7D C4 7B C4 7B E0 7B
7B39 = F0 7B FC 7B FF 7C 03 7B
7B41 = F6 7C 63 7C 68 7C 73 7C
7B49 = 86 7D 00 7D 3D 7B B7 20
7B51 = 70 79 FE EE 20 0F 2A 1C
7B59 = 7F 23 46 23 4E 22 1C 7F
7B61 = ED 43 1E 7F C9 FE AE 38
7B69 = 09 20 2C 2A 1C 7F 22 10
7B71 = 7F C9 FE A8 20 0F 2A 1C
7B79 = 7F 23 46 23 4E ED 43 10
7B81 = 7F 22 1C 7F C9 FE A0 C0
7B89 = 2A 1C 7F ED 5B 10 7F 73
7B91 = 2B 72 2B 22 1C 7F C9 FE
7B99 = E0 20 13 21 00 70 11 01
7BA1 = 70 75 01 FF 07 ED B0 3A
7BA9 = 3B 78 32 00 68 C9 E6 F0
7BB1 = FE C0 C0 79 17 17 17 17
7BB9 = E6 10 C6 09 32 3B 78 18
7BC1 = E6 C5 C9 2A 1C 7F ED 5B
7BC9 = 1E 7F 73 2B 72 2B 22 1C
7BD1 = 7F 18 8D 1A B9 C0 2A 1E
7BD9 = 7F 23 23 22 1E 7F C9 1A
7BE1 = B9 C8 18 F2 79 1F 1F 1F
7BE9 = 1F E6 0F 6F 26 7F C9 CD
7BF1 = E5 7B 4E 18 DE CD E5 7B
7BF9 = 4E 18 E4 79 12 C9 1A 81
7C01 = 12 C9 CD E5 7B 79 E6 0F
7C09 = FE 06 28 2F 30 47 FE 03
7C11 = 28 13 30 15 B7 20 03 7E
7C19 = 12 C9 3D 20 04 1A B6 12
7C21 = C9 1A A6 12 C9 1A AE 12
7C29 = C9 FE 04 20 0A 1A 86 12
7C31 = 3E 00 8F 32 0F 7F C9 1A
7C39 = 96 18 F4 D5 4E 1A 5F 06
7C41 = 08 16 00 62 6A 29 CB 11
7C49 = 30 01 19 10 F8 D1 7D 12
7C51 = 7C 32 0F 7F C9 FE 0D CA
7C59 = 34 7E FE 0E CA 78 7E C9
7C61 = 00 00 ED 43 10 7F C9 3A
7C69 = 00 7F 6F 26 00 09 22 1E
7C71 = 7F C9 21 20 7F 34 6E 26
7C79 = 24 3A 21 7F 86 2B AE 32
7C81 = 21 7F A1 12 C9 79 E6 0F
7C89 = B7 20 02 3E 10 D9 47 D9
7C91 = CD E5 7B 7E 26 00 87 87
7C99 = 6F 29 29 29 44 4D 1A E6
7CA1 = 03 D9 CD 70 7E 5F 08 D9
7CA9 = AF 32 0F 7F 2A 10 7F 56
7CB1 = 23 5E 23 E5 2E 00 79 87
7CB9 = 28 09 CB 3A CB 1B CB 1D
7CC1 = 3D 20 F7 7A CD E4 7C 7B
7CC9 = CD E4 7C 7D CD E4 7C D9
7CD1 = 79 C6 20 4F 78 CE 00 E6
7CD9 = 07 47 08 5F 08 D9 E1 10
7CE1 = CE D9 C9 D9 B7 28 11 60
7CE9 = 69 16 70 19 57 AE 77 A2
7CF1 = BA 28 05 3E 01 32 0F 7F
7CF9 = 7B 3C E6 1F 5F D9 C9 79
7D01 = FE B3 28 19 30 1E D9 CD
7D09 = F4 2E D9 47 1A B8 79 28
7D11 = 06 FE A1 C0 C3 D7 7B FE
7D19 = 9E C0 C3 D7 7B D9 CD F4
7D21 = 2E D9 12 C9 DB 20 06 05
7D29 = 1F 30 02 10 FB 3E 37 80
7D31 = 6F 26 7D 7E 12 C9 00 0D
7D39 = 2C 4D 20 2E 79 FE 29 28
7D41 = 48 30 44 FE 18 28 44 30
7D49 = 51 FE 02 20 09 1A 6F 26
7D51 = 00 23 22 96 7D C9 FE 0A
7D59 = 20 1B D9 CD F4 2E B7 20
7D61 = FA CD F4 2E B7 28 FA CD
7D69 = F4 2E B7 28 F4 08 CD 50
7D71 = 34 08 D9 12 C9 21 FE 8A
7D79 = 7E FE E5 20 04 23 7E FE
7D81 = 8B C2 E9 7A C3 FD 8A 18
7D89 = 65 18 42 1A 6F 26 00 29
7D91 = 29 23 4D 44 21 2D 00 C3
7D99 = 5C 34 FE 1E 20 0C 2A 10
7DA1 = 7F 1A 4F 06 00 09 22 10
7DA9 = 7F C9 1A 6F D9 16 21 5A
7DB1 = 4A D9 3A 3B 78 57 0E 10
7DB9 = D9 CD 73 7C D9 AA 57 32
7DC1 = 00 68 06 70 10 FE 0D 20
7DC9 = EF 2D 20 EA C9 1A E6 0F
7DD1 = 47 87 87 80 C6 30 5F 16
7DD9 = 7F 0E 05 41 21 12 7F 22
7DE1 = 10 7F 1A E6 FF 77 23 13
7DE9 = 36 00 23 10 F5 C9 FE 65
7DF1 = 28 2A 30 20 FE 33 20 2F
7DF9 = 1A 2A 10 7F 06 64 CD 09
7E01 = 7E 06 0A CD 09 7E 77 C9
7E09 = 0E 00 18 02 0C 90 B8 30
7E11 = FB 71 23 C9 1A 32 E5 7D
7E19 = 32 5F 7E C9 1C 4B 06 00
7E21 = 58 2A 10 7F C3 F5 7E 1C
7E29 = 4B 06 00 58 2A 10 7F EB
7E31 = C3 28 7F 1A 4F 46 AF 32
7E39 = 0F 7F CB 79 C0 78 FE 40
7E41 = D0 87 87 6F 26 00 29 29
7E49 = 29 79 51 00 1F 1F E6 1F
7E51 = 4F 06 70 09 7A E6 03 C6
7E59 = 6C 5F 16 7E 1A E6 FF 57
7E61 = AE 77 A2 BA C8 3E 01 32
7E69 = 0F 7F C9 C0 30 0C 03 4F
7E71 = D9 1A 1F 1F E6 1F C9 1A
7E79 = 4F 46 3A 0F 7F 5F DD 2A
7E81 = 10 7F AF 32 0F 7F 18 11
7E89 = D5 7B 08 78 84 47 79 85
7E91 = 4F 08 3D 20 F5 15 20 F1
7E99 = D1 CD CB 7E C8 CB 7F 28
7EA1 = E7 D5 7B 08 C5 D9 C1 CD
7EA9 = 3B 7E D9 78 84 47 79 85
7EB1 = 4F 08 3D 20 EE 15 20 EA
7EB9 = D1 CD CB 7E C8 CB 7F 20
7EC1 = E0 C5 D9 C1 CD 3B 7E D9
7EC9 = 18 BE 21 00 00 DD 7E 00
7ED1 = B7 C8 E6 07 20 02 3E 08
7ED9 = 57 DD 7E 00 DD 23 CB 77
7EE1 = 28 01 2D CB 6F 28 01 2C
7EE9 = CB 67 28 01 24 CB 5F 28
7EF1 = 01 25 B7 C9 ED B0 22 10
7EF9 = 7F C9 00 00 00 00 00 11
7F01 = 11 11 11 11 11 11 11 11
7F09 = 11 11 11 11 11 11 11 11
7F11 = 11 11 11 11 11 11 11 11
7F19 = 11 11 11 11 11 11 11 11
7F21 = 11 00 00 00 00 00 00 ED
7F29 = B0 EB 22 10 7F C9 00 FC
7F31 = CC CC CC FC 30 30 30 30
7F39 = 30 FC 0C FC C0 FC FC 0C
7F41 = FC 0C FC C0 C0 CC FC 0C
7F49 = FC C0 FC 0C FC FC C0 FC
7F51 = CC FC FC 0C 0C 0C 0C FC
7F59 = CC FC CC FC FC CC FC 0C
7F61 = FC FC CC FC CC CC F0 CC
7F69 = FC CC F0 FC C0 C0 C0 FC
7F71 = F0 CC CC CC F0 FC C0 FC
7F79 = C0 FC FC C0 F0 C0 C0 00
7F81 = 00 0
```

PLOT RIGHT 1 DOT, PLOT DOWN 1 DOT, PLOT LEFT 1 DOT, PLOT UP 1 DOT, END. From Figure 2, the codes are: 1 0 1 0 0 0 0 1, 1 0 0 1 0 0 0 1, 1 1 0 0 0 0 0 1, 1 0 0 0 1 0 0 1, '00'. That is: A1 91 C1 89 00 in hex. The program shown in Listing 2e uses this data to draw squares of random sizes all over the screen — try it!

Using sound commands

Table 3 shows pitch and duration values used in VZ/Chip-8 sound commands. The values given here are not tuned to a standard pitch, but are chosen so that the scale sounds reasonably tuneful when played.

To play a note, of duration V1, at pitch V2, use a segment of code like: F292 F118. Be sure to use the correct duration for the pitch under consideration, otherwise your tunes will sound uneven! You

LISTING 2a.

```
8200 — 6A 00 — put '00' to VA
       A2 0A — point I at 820A, the start of the shape data
       DA A9 — show a nine row shape at (VA,VA) ie (0,0)
       FB 0A — wait for a key to be pressed, store its value in VB
       F0 00 — end
820A — 0A A8  }
       09 98  }
       0A A8  }
       00 C0  }
       03 FO  }   data for the shape in Figure 1.
       3C CF  }
       00 C0  }
       03 30  }
       0C 0C  }
```

LISTING 2b. RANDOM DOTS

```
8200 — CA 7F — put a random number (less than 7F) to VA
       CB 3F — put a random number (less than 3F) to VB
       CC FF — put a random number in VC
       FC CC — load the colour register with VC (ie: random colours)
       8A BD — plot a point at (VA,VB), a random screen position
       EF B3 — scan the keyboard and load the key pressed into VF
       3F 01 — if that key is '01' (the BREAK key), skip the next instruction
       12 00 — otherwise, go back to the start (plot another point)
       F0 00 — end; if BREAK key is down, the program will end
```

LISTING 2c. SCREEN FULL O' NUMBERS

```
8200 — 6F AA
       FF CC — load colour register with blue
       6A 00 — '00' to VA
       6B 00 — '00' to VB
8208 — 6C 00 — '00' to VC
820A — FC 29 — prepare to show VC as a number
       DA B5 — show the number at (VA,VB)
       7A 08 — increase VA by '08', the next number will be beside the one just shown
       7C 01 — increase VC by '01', the next number to display is one more than the last
       3C 10 — if the whole row has been shown, skip next instruction
       12 0A — otherwise, go back to 820A and show another number
       7B 08 — prepare to show on next row; increase VB by '08'
       3B 40 — if we have finished the last row, skip next instruction
       12 08 — otherwise, go back to 8208, begin a new row
       FF 0A — full screen; wait for a key to be pressed
       F0 00 — end
```

LISTING 2d. COUNTING

```
8200 — 6F FF FF CC 6A 00 22 28 6B 00 6C 00 7C 01 3C 00
8210 — 12 0C 7B 01 3B 06 12 0A 22 28 7A 01 4A 64 6A 00
8220 — EF B3 3F 01 12 06 F0 00 A2 40 FA 33 A2 40 F2 65
8230 — 6B 00 6C 00 F1 29 DB C5 7B 04 F2 29 DB C5 00 EE
8240 — 00 00 00 00
```

LISTING 2e. LOTS OF SQUARES

```
8200 — 65 FF F5 CC 6A 00 C6 7F C7 3F C5 1F 86 55 87 55
8210 — 85 54 75 01 8F 50 A2 24 86 7E 7A 01 3A 20 12 06
8220 — FF 0A F0 00 A1 91 C1 89 00 00
```

LISTING 2f. CHIRP

```
8200 — CE 07 7E 02 CA 0F FA 02 FE 18 7A 01 3A 18 12 06
8210 — EF B3 3F 01 12 00 F0 00
```

don't have to stick to the pitch and duration values shown in Table 3, so other effects, such as sirens, can be created. A sample sound program is shown in Listing 2f.

Saving completed programs

When you have written a program, and are satisfied that it does what you want, save it. There are two options here:

(i) Save the program *with* the editor. This is for programs which still have not been fully finished. Save all memory from 7AE9 to 8F30.

(ii) Save the program *without* the editor. This is for complete programs, only save memory from 7AE9 to the end of your Chip-8 program.

In either of the above cases, tape users will have to put up with the program running whenever it is loaded, so if the program is incomplete, make sure it ends otherwise you will never be able to edit it!

NOTE: We have had complaints from readers who could not get the editor listed last month running. Printed below are corrections to lines 70 and 380, and two new lines 770, 780 to be added. As well as this, we understand that in some issues of the magazine, the figure 32 between 90 and D6 in line 510 was printed so indistinctly as to look like 37. So if you have any problems after amending the listing, check line 510.

CORRECTIONS TO THE 'EDITOR' LISTING.
THE FOLLOWING ARE THE CORRECTED LINES.

```
70 IFT = 118550, PRINTUSR (1)

380 DATA10,78,C9,DF,20,E9,F1,11,
27,8E,C3,7B,8B,00,00,00

770 DATA8B,C3,Ec,8B,2D,22,A0,78,
C9,00,00,00
780DATAZZ
```

NB. THE LAST TWO LINES NEED TO BE ADDED TO THE PROGRAM.

Those who couldn't be bothered typing in Listing 1 can get a copy (tape only) by writing to 'Chris Griffin, PO Box 233, Diamond Creek, Victoria 3089' and including $5 with the letter (for postage, packing, tape, and my time!).